SER 502 Project Milestone 1

Team 16
Spring 2023
Sudheer Reddy Kunduru
Swapnil Mukeshbhai Chadotra
Jay Yogeshbhai Patel
Manan Hasmukhkumar Patel
Jashmin Mineshkumar Patel

Name of Programming Language	Pixel
File Extension for programs	.pixel
Programming Language Paradigm	Imperative Programming
Github Link to the code	https://github.com/sud2701/SER502-Spring2023-Team16

Source Program

All the programs developed using the Pixel language will have the .pixel file extension and begin with main function/block.

Lexical Analysis

This phase takes the program file as input and generates a series of tokens. Lexical analysis begins by deleting any undesired things from the program such as comments, white spaces etc. Since the programs written using pixel will span multiple lines of code, the new line character (\n) will be used for breaking down the program into smaller chunks or instructions. Then within each line, each keyword or identifier or parenthesis or value will be tokenized separately. Lexical Analysis will be performed using Python.

Parsing

This phase takes the series of tokens generated by the lexical analyzer as input and validates whether the program is syntactically correct. Parsing will be performed using Prolog through

pattern matching on a Definite Clause Grammar (DCG) that represents Pixel's syntax. The parser will make use of the List data structure which is well handled using prolog's predicates. The output of this phase will be a parse tree.

Semantic Evaluation / Interpretation

This phase involves the execution of the program line by line and the evaluation of semantics within each line such as the correctness of used operators, evaluation of expressions, variable scoping, etc. Evaluation will be performed using prolog and a set of environment variables.

Tools to be used

Python - For Lexical Analysis
Prolog - For Generation of Parse Trees
Prolog - For Semantic Evaluation
Data Structures to be used - Lists

Language Design

1. Every program begins with a main block
2. Within the main block, a set of declarations will be followed by a set of commands/control structures
3. The declarations will be separated from each other by a semi-colon
4. Three datatypes are allowed: integer, boolean, string
5. Values of type integer can be either positive integers or negative integers or zero.
6. Entities of type boolean will assume one of the two values: true, false
7. Values of string type will begin and end with single quotes(‘) and can contain a combination of lower case letters, upper case letters and numbers of any length and order in between the quotes. Empty strings are supported and are represented as ‘’
8. The set of declarations will be separated from the set of commands using a semi-colon as well
9. Two types of declarations are allowed. They are Declaration of constant entities and declaration of variables
10. A constant declaration uses the keyword “const” followed by its datatype, then an identifier for the constant, the assignment operator (=) and the value of the declared constant. The value of a constant should be of the specified datatype and is not allowed to be changed.

11. A variable declaration is similar to that of a constant declaration but uses the “var” keyword instead of “const” and the remainder of the declaration is similar. However, the value of a variable can be changed as many times as possible.
12. Pixel is a strictly typed language. To reduce the complexity associated with type conversion, changing the datatype of a variable is not allowed. Once a datatype is declared for a variable/constant, it can’t be changed.
13. Identifiers for variables/constants can be any combination of lower case and upper case letters of any length starting from 1.
14. Assignment expressions begin with an identifier, followed by the assignment operator (=), the value to be assigned, and end with a semi-colon. Assignment expressions are intended to be used for assigning values to variables that are already declared.
15. Five types of operators are supported. They are
    a. **Arithmetic operators**

       +, - , *, /

    b. **Relational operators**

       >, <, >=, <=, ==

       > - Greater than

       < - Less than

       >= - Greater than or equal to

       <= - Less than or equal to

       == - Comparison of values for equality

    c. **Boolean operators**

       and - Represents Logical AND operation

       or - Represents Logical OR operation

       not - Represents Logical NOT operation

    d. **Ternary operator**

       ? : - Evaluates a boolean expression to decide which command to execute among the two specified commands

    e. **Assignment operator**

       = - used for assigning values to variables/constants

16. Three types of control structures are supported. They are

    a. **If-else block**

Three flavors of if-else are supported. They are:

Regular if - else

```
if ( <condition>) {
        // Commands
}
else {
        // Commands
}
```

Else - if ladder without an else block

```
if ( <condition 1>) {
        // Commands
}
else if (<condition 2>) {
        // Commands
}
else if (<condition 3>) {
        // Commands
}
……..
else if (<condition n>) {
        // Commands
}
```

Else - if ladder with an else block

```
if (<condition 1>) {
        // Commands
}
else if (<condition 2>) {
        // Commands
}
else if (<condition 3>) {
        // Commands
}
```

```
……..
else if (<condition n>) {
        // Commands
}
else {
        // Commands
}
```

b. For loop

For loop is intended for performing iteration when the number of iterations to be performed is known.

Four flavors of for loop are supported. They are:

1. **Regular for loop**

```
for ( <declaration> ; <condition> ; <assignment> ) {
        // Commands
}
```

<declaration> - for creating a variable used for iteration

<condition> - boolean expression that evaluates to true or false

<assignment> - for updating the iteration variable

Commands - series of commands to be executed repeatedly until the condition returns false

Example:

```
for ( var integer a = 5 ; a <= 10 ; a = a + 2 ) {
        print(a);
}
```

Output - 5 7 9

2. **For loop with assignment**

```
for ( <assignment> ; <condition> ; <assignment> ) {
        // Commands
}
```

Similar to the regular for loop but has as assignment instead of a declaration

Intended for utilizing an existing variable for iteration instead of creating a new variable

Example:

```
var integer a = 0;
for ( a = 5; a <= 10 ; a = a + 2 ) {
        print(a);
}
```

Output - 5 7 9

3. **For loop within a range with start, end and increment values**

```
for <identifier> in range ( <startvalue>, <endvalue>, <increment> ) {
        // Commands
}
```

<identifier> - identifier for the integer variable to be used for iteration
<startvalue> - starting value for the variable
<endvalue> - ending value for the variable, the loop terminates when the value of the variable exceeds this value
<increment> - the amount by which the value of the variable gets incremented after every iteration
Commands - series of commands to be executed in every iteration

Example:

```
for a in range ( 5, 10, 2 ) {
        print(a);
}
```

Output - 5 7 9

4. **For loop within a range with just the end value**

```
for <identifier> in range ( <endvalue> ) {
        // Commands
}
```

<identifier> - the identifier of the variable to be used for iteration
Here, the value of the variable begins at 0 and will be incremented by 1 in every iteration until the value exceeds <endvalue>
Commands - series of commands to be executed in every iteration

Example:

```
for a in range ( 10 ) {
```

```
        print(a);
    }
```

Output - 0 1 2 3 4 5 6 7 8 9 10

c. While loop

```
while ( <condition> ) {
    // Commands
}
```

<condition> - boolean expression used for terminating the loop
When <condition> evaluates to false, the loop will terminate
Commands - series of commands/instructions to be executed in every iteration

17. Each of the commands within a control structure is either an assignment or a ternary operation or a print statement or another control structure. Control structures begin and end with square parenthesis ('{' and '}'). Assignments, ternary operations and print statements end with a semi-colon (';').

18. A print statement is written as

```
print ( <argument> ) ;
```

Where <argument> can be an integer value, a string value, a boolean value, an identifier for a variable/constant or any kind of expression (relational, arithmetic or boolean). If the <argument> is a value, the statement just prints the value as output. If the argument is an identifier, then the value associated with the identifier will be printed as output. However, if the argument is an expression, the expression will be evaluated and the resulting value will be printed as output.

Language Grammar

Program - PROGRAM
Main Block - MAIN
Declaration - DEC
Command - CMD
Constant Declaration - CNT
Variable Declaration - VAR
Expression - EXPR
Boolean expression - BOOL

Relational Expression - REL
Ternary Operation - TER
Assignment - ASG
Else-if ladder - ELIF
Number - NUM
Identifier - ID
Lower Case Letter - LL
Upper Case Letter - UL
Positive Integer - POSITIVEINTEGER
Negative Integer - NEGATIVEINTEGER
Integer - INT
String - STR
Comment - CMT

PROGRAM ::= main () { MAIN }
MAIN ::= DEC ; CMD
DEC ::= DEC ; DEC | CNT | VAR
CNT ::= const integer ID = INT | const integer ID = EXPR | const boolean ID = BOOL | const string ID = STR
VAR ::= var integer ID = INT | var integer ID = EXPR | var boolean ID = BOOL | var string ID = STR
BOOL ::= BOOL and BOOL | BOOL or BOOL | not BOOL | REL | true | false
EXPR ::= EXPR + EXPR | EXPR - EXPR | EXPR * EXPR | EXPR / EXPR | (EXPR) | ID | NUM
REL ::= EXPR > EXPR | EXPR < EXPR | EXPR >= EXPR | EXPR <= EXPR | EXPR == EXPR
ID ::= LL ID | UL ID | LL | UL
STR ::= ‘LL STR’ | ‘UL STR’ | ‘N STR’ | ‘N’ | ‘LL’ | ‘UL’ | ‘’
POSITIVEINTEGER ::= NUM NUM | NUM
NEGATIVEINTEGER ::= - POSITIVEINTEGER
INT ::= POSITIVEINTEGER | NEGATIVEINTEGER
TER ::= BOOL ? CMD : CMD
ASG ::= ID = EXPR

CMD ::= CMD CMD |
if (BOOL) { CMD } else { CMD } |
if (BOOL) { CMD } ELIF |
if (BOOL) { CMD } ELIF else { CMD } |
for (DEC ; BOOL ; ASG) { CMD } |
for (ASG ; BOOL ; ASG) { CMD } |

for ID in range (INT, INT, INT) { CMD } |
for ID in range (INT) { CMD } |
while (BOOL) { CMD } |
ASG; |
TER; |
print (EXPR); |
print (STR); |
print (BOOL); |
print (INT);

ELIF ::= else if (BOOL) { CMD } ELIF | else if (BOOL) { CMD }
CMT ::= // STR
N ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
LL ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
UL ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z